Exporting Databases in XML
A Conceptual and Generic Approach

Ph. Thiran[1], F. Estiévenart[2], J-L. Hainaut[3], and G-J. Houben[1]

[1] Technische Universiteit Eindhoven, The Netherlands
[2] CETIC Research Center, Gosselies, Belgium
[3] Université de Namur, Belgium

Abstract. This paper describes a generic conceptual framework in which semantics-based XML DTD can be derived from existing, and more generally legacy, databases. It consists in first recovering the conceptual schema of the database through reverse engineering techniques, then in converting this schema, or part of it, into XML-compliant data structures. Both steps heavily rely on generic schema transformation techniques, while all the schemas involved in the whole process are expressed in a unique model, named GER. This approach is supported by a CASE environment.

1 Introduction

The quantity of information available on the World Wide Web continuously increases. Most of this information is provided by Web information systems that combine traditional storage mechanisms, e.g. relational databases, with the easy access mechanism that gave the Web its popularity. The term *deep Web* is sometimes used to distinguish the wealth of information stored in these Web-accessible information systems from the *surface Web* of explicitly linked HTML pages. However, the information on the (deep) Web is placed there independently by different organizations. As a consequence, data sources containing related information can appear at different Websites and systems, in different formats, and for different purposes. Moreover, the current Web information systems are lacking particularly in the area of semantics, for example in terms of relationships within the data.

Currently, XML is becoming the *de facto* standard for publishing and exchanging data over the Web. The use of XML as the common format for representing, exchanging, storing, and accessing data poses new challenges to data systems. Since the majority of everyday data is still stored and maintained in *standard* database systems, we expect that the needs to export stored data in XML format will grow substantially. To this end, several projects recently investigated the issues of converting database schema in XML ([1], [2], [3], [4], [5]).

In this paper, we focus on exporting database schemas into XML DTD [6]. As mentioned, some research projects have already investigated issues related to schema conversions between XML and database models. Unlike their approaches, we investigate the problem from a *generic* and *conceptual* perspective:

– *Generic perspective.* Current approaches for exporting databases into XML rely on couples of models, such as those intended to produce XML views of relational schemas. In this work, we use a high-level formalism - named Generic Entity Relationship (GER) - able to express schemas whatever their underlying data model and their abstraction level. Such a formalism defines a reference model on which transformational operators are built. That allows the description of arbitrary models (including DTD) and the use of schema conversions between two not necessarily equivalent models (*-to-DTD). As a result, our approach is not limited to any specific structured data model.
– *Conceptual perspective.* Most current transformation strategies consist in translating each construct of the source database into the closest constructs of XML without attempting any semantic interpretation. They capture the structure of the original schema and largely ignore the interconnection among data and the hidden semantic constraints. Our strategy consists in recovering the precise semantic description (i.e., its conceptual schema where multiple entities are interconnected) of the source database first, through reverse engineering techniques, then in developing the DTD from this schema through a semi-automated model translation (i.e., converting multiple inter-connected entities into a coherent and hierarchical parent-child structure).

To focus the presentation on the translation problem, the paper has been written for a scenario in which the whole database is to be translated into XML structures. In actual situations, only selected parts have to be exported or transformed. Various practical scenarios can easily be derived from the proposed framework.

This paper is organized as follows. In section 2, we present our previous work to show how a DTD can be represented in a generic entity-relationship model, the GER. This leads to the first technical contribution of the paper – providing a high-level reference model for database and XML models, and hence, the possibility of transforming data instances between them. In Section 3, we develop our conceptual approach for converting databases into XML DTD. The approach is illustrated by a case study that points at different benefits of conversion. This leads to the second technical contribution of the paper – providing a method for extracting the DTD of databases which captures their hidden semantic structures. Section 4 presents an operational CASE tool – DB-MAIN – that supports our approach. Finally, Section 5 concludes this paper.

2 Representing XML DTD in the GER

In [7], we showed how a wide spectrum entity-relationship model, the GER can express data structure schemas whatever their underlying data model and their abstraction level. Moreover, we also showed that it is possible to transform the constructs from one data model into those of another model. In this section, we extend our work and specify how XML DTD can be represented in the GER. As such, we are adding XML DTD to the set of operational models whose schemas can be transformed into each other using our generic framework. For reasons of space, we refer the reader to our earlier work [7] for a full definition of the GER and the schema transformations.

2.1 DTD expressed in terms of GER

Expressing XML data in terms of a reference model has already received much attention and the common approach is to use some variant of entity-relationship model, such as UML structure diagrams, for representing XML or other kinds of semi-structured data ([8] and [9]). However, due to the differences between ER (or UML) and XML models, this approach cannot address all the DTD notions (e.g., data order or `ID/IDREF` attribute). We use a high-level data model (GER) to represent XML DTD. The GER model supports, in an abstract form, such notions as `IDREF` attributes, ordering information or alternate structures found in XML documents.

Different frameworks for describing XML documents have been proposed. The first and most widely used method in real-world documents is the Document Type Definition (DTD), thus we have based our approach on DTD. The most prominent alternative description mechanisms is XML Schema. The latter is a more powerful model that can capture advanced constructs as different types of relationships, precise cardinalities, inheritance and attribute type. Though the GER can describe the concepts of XML Schema as well, we have chosen to limit the representation to DTD, both for scope limit and due to its wider popularity so far.

For the needs of this paper, the GER can be perceived as an enriched variant of the standard entity-relationship model. It includes the concepts of *entity type*, *attribute*, *value domain* and *relationship type*. Attributes can be atomic or compound, mandatory or optional, single-valued ou multivalued. The roles of a relationship type can be labelled; it has a cardinality constraint (a pair of integers stating the range of the number of relationships in which any entity can appear). An attribute has a cardinality constraint too, that states how many values can be associated with each parent instance (default is `1-1` and does not appear in graphical schemas). In general, several properties hold, and must be declared, among the components of an entity type: uniqueness, referential and existence constraints are just some of them. Due the wide variety of such properties, the GER include the generic concept of property *group*, or group for short. A group is any subset of components (attributes and/or roles) of an entity type on which one or several properties are defined. The label(s) of the group specifies its properties (`id` for identifier, `ref` for referential, `excl` for exclusive, and so on). For example, a group of attributes of entity type `E` can be declared identifier and referential. This group models such relational pattern as a primary key that simultaneously is a foreign key. In DTD representation, property groups will be used to model `ID`, `IDREF`, sequence, choice. However, they can also model constraints that cannot be explicitly expressed in DTD, and that will be translated in another way (though filters or procedural components for example).

In this section, we describe how any DTD document (`DTDd`) can be expressed as an equivalent GER schema (`DTDs`). We successively explain how the GER model can represent the concepts of (1) *element types*, (2) *content types* and (3) *attribute types*. Table 1 summarizes the correspondences between the main DTD concepts and their GER interpretation. Our development is illustrated through a concise `DTDd` describing `Orders` and their related `Products`. Figure 1 shows the `DTDd` and the resulting *equivalent* `DTDs` expressed in terms of GER.

Table 1. DTD concepts expressed in terms of GER.

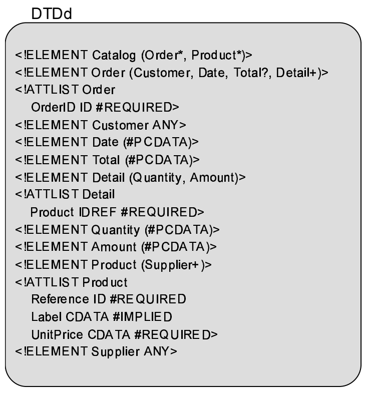
DTD concept	GER construct
Element type	Entity type
Content type `PCDATA/ANY`	Attribute `#pcdata/#any`
Content type `ELEMENT`	Relationship type
Occurence operators	Role cardinalities
Sequence/Choice organisation	`seq/choice` group
Attribute type	Attribute
Attibute modifiers	Attribute cardinalities
`ID/IDREF` attribute	`gid/idref` group

Element Type Definition Each element type declared in `DTDd` (including the root element) is expressed by an entity type in `DTDs`, the name of which is that of the element.

Content Type Definition. According to the `DTD` conventions, a content type is either `PCDATA`, `ANY`, `EMPTY` or `ELEMENT`. In `DTDs`, the content type directly determines the properties (attributes, roles...) of an entity type.

Content types for leaf elements. Content type `PCDATA` is associated to leaf elements that are expected to only contain simple-text values. In `DTDs`, such elements are represented by an entity type having as unique attribute `#pcdata`. The content type `ANY` is similarly modelled by a simple atomic attribute named `#any` (by convention). Contrary to the content types `PCDATA` and `ANY`, no special attribute is needed to represent an `EMPTY` element. The absence of specific attributes (`#pcdata` or `#any`) is simply used to represent the content type `EMPTY`.

Example. In Figure 1, the entity types `Date`, `Total`, `Quantity` and `Amount` represent leaf elements with content type `PCDATA`. By analogy, both the entity types `Customer` and `Supplier` express elements with content type `ANY`.



DTDd

```
<!ELEMENT Catalog (Order*, Product*)>
<!ELEMENT Order (Customer, Date, Total?, Detail+)>
<!ATTLIST Order
   OrderID ID #REQUIRED>
<!ELEMENT Customer ANY>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Total (#PCDATA)>
<!ELEMENT Detail (Quantity, Amount)>
<!ATTLIST Detail
   Product IDREF #REQUIRED>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT Amount (#PCDATA)>
<!ELEMENT Product (Supplier+)>
<!ATTLIST Product
   Reference ID #REQUIRED
   Label CDATA #IMPLIED
   UnitPrice CDATA #REQUIRED>
<!ELEMENT Supplier ANY>
```

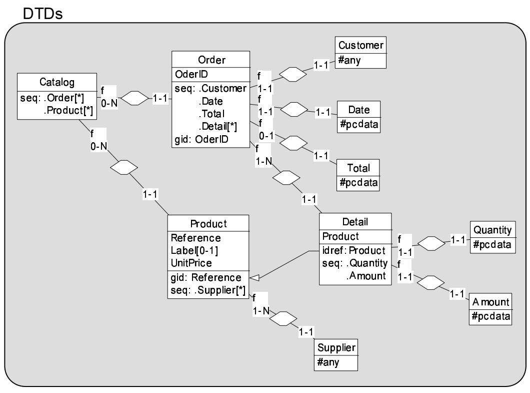


Fig. 1. The `DTDd` and `DTDs` of `Catalog`.

Content types for non-leaf elements. An element type that is not a leaf element is inevitably composed of one or more subelements. The hierarchical relationship between two elements is materialized, in `DTDs`, by a one-to-many binary relationship type (called *hierarchical relationship type*) binding the father to the child entity types. The role played by the father entity type is called *father role* (in opposition with *child role*) and is named `f` (for **f**ather).

It is important to notice that, due to the hierarchical organization of XML elements, the structure formed by the entity types and the relationship types has to respect three basic properties:

– *Uniqueness of the root.* A `DTDs` always possesses one (and only one) particular entity type that has no father; it is called the *root entity type.*
– *Uniqueness of fathers.* Any entity type must play one, and only one, child role in `DTDs`[4].
– *Absence of cycles.* In an oriented graph, a cycle occurs when a node belongs to the list of its own descendants. In a schema made of entity types (nodes) and directed hierarchical relationship types (arcs), cycles are prohibited in order to reflect a hierarchical structure.

The DTD conventions allow the definition of occurrence operators on subelements. `?`, `*` and `+` stand respectively for `0-1`, `0-N`, `1-N` cardinalities. The absence of such operator is equivalent to cardinality `1-1`. In `DTDs`, the minimum and maximum cardinalities of a father role are used to express that kind of information.

If an element is composed of several subelements, the latter can be organized in a sequence structure. In this specific case, the father element must hold a

[4] Except for the root entity type that only plays father role(s). In addition, the fact that an element appears in more than one father element cannot be represented by more than one relationship type for reasons that are beyond the scope of this paper.

group named `seq`. That group contains the child roles played by all the subelements. Note that the order of the components within a `seq` is meaningful as it specifies the order of the elements in the sequence.

Example. In Figure 1, the entity type `Order` is composed of a sequence of four children (respectively entity types `Customer`, `Date`, `Total` and `Detail`). The father roles played by the entity type `Order` are named `f` and have corresponding cardinalities (respectively `1-1`, `1-1`, `0-1` and `1-N`).

Attribute Type Definition. An attribute type declared within an element type appears in a `DTDs` in the form of an attribute in the corresponding entity type.

Attribute modifiers. In `DTDs`, the default cardinality for an attribute is `1-1`, meaning that the attribute is single-valued and mandatory (keyword `#REQUIRED` in the DTD conventions). Declaring an optional attribute (keyword `#IMPLIED`) simply consists in setting its minimal cardinality to `0`. The only way to express multiple values for a same information is to represent this one by an element and an occurence operator such as `*` or `+`. Multivalued attributes are therefore **not** allowed in `DTDs`.

`ID` attributes. Contrary to a common identifier[5] whose scope is restricted to the entity type in which it is declared, an `ID` attribute in XML has a general scope extended to the entire document. However, as they define similar constraints of uniqueness, XML `ID` attributes have been assigned a notation close to the one used for common identifiers (though things have improved in XML Schemas). Indeed, they must belong to a group labelled, by convention, `gid` (standing for **g**lobal **id**entifier). Besides their difference of scope, `id` and `gid` groups present another noticeable difference: an `id` group is allowed to include several attributes or roles while a `gid` group comprises one attribute only.

`IDREF` attributes. `IDREF` attributes implement a referential mechanism that we could compare to traditional foreign keys with a general scope. The conventions used to represent foreign keys and `IDREF` attributes are analogous. An `IDREF` attribute is component of a group `idref`.

Example. In the entity type `Product` (Figure 1), `UnitPrice` is a simple monovalued and mandatory attribute while `Label` is optional (cardinalities `0-1`). `Reference` is an attribute contained in a `gid` group to reflect its XML `ID` type.

2.2 Transforming XML DTD in GER

In [7], we define a set of transformations for transforming a GER schema. These transformations can be applied by a user to map between schemas expressed in

[5] group labelled `id` in the GER.

the same or different data modelling languages. The use of GER as the unifying data model allows constructs from different data modelling languages to be mixed in the same intermediate schema.

A transformation consists in deriving a target schema `S'` from a source schema `S` by replacing construct `C` (possibly empty) in `S` with a new construct `C'` (possibly empty). More formally, a transformation Σ can be completely defined by a pair of mappings `<T,t>`: `C' = T(C)` and `c' = t(c)`. `T` is the structural mapping, that explains how to replace construct `C` with construct `C'` while `t`, the instance mapping, states how to compute instance `c'` of `C'` from any instance `c` of `C`.

Any transformation Σ can be given an inverse transformation Σ_i, such that $\mathtt{T}_i\mathtt{(T(C))} = \mathtt{C}$. If, in addition, we also have: $\mathtt{t}_i\mathtt{(t(c))} = \mathtt{c}$, then Σ is declared semantics-preserving.

A transformation sequence is a sequence of $\mathtt{n} \geq \mathtt{1}$ primitive transformations: `Tp = <T1 T2 ... Tn>`. For instance, `Tp = <T1 T2>` is obtained by applying `T2` on the schema that results from the application of `T1`.

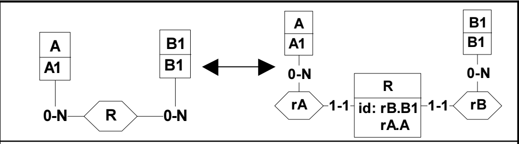
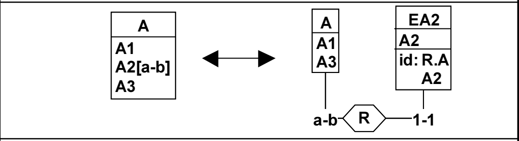
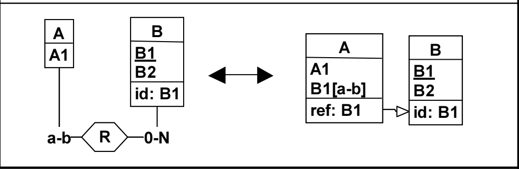
Transformations are operators that manipulate GER constructs. We propose in Figure 2 two sets of the transformational operators used in this paper (Section 3). The first one is made up of three standard transformations used in translation of relational schemas into conceptual schemas (named hereafter *standard* transformations). The second comprises additional techniques suited to derive a DTD from a structured data schema (named hereafter *DTD-specific* transformations). These transformation operators are formally described in [10] and in [11].

3 Database Exportation in XML

This section describes a conceptual approach for converting databases into DTD. Normally, a precise and detailed documentation should be associated with any database, in such a way that the DTD expression should be straightforward. However, the actual state of most databases can be described, at best, as undocumented. Hence the need to rebuild the conceptual schema of the source database when it is no longer available. This process is commonly called *reverse engineering.*

Our strategy consists in recovering the precise semantic description of the source database before converting it into DTD. It follows a semi-automated four-step method (Figure 3):

– *Schema extraction:* The physical schema (`PS`) of the database is expressed in GER.
– *Schema conceptualization:* The conceptual schema (`CS`) is recovered from the physical schema (`PS`).
– *Model translation:* The GER DTD schema (`DTDs`) is semi-automatically generated from the conceptual schema (`CS`) by means of schema transformations.
– *Schema exportation:* The DTD schema (`DTDs`) is expressed as a DTD document (`DTDd`).

RT-ET: Transforming a relationship type into an entity type. *Inverse:* **ET-RT**	 A A1 B1 B1 0-N R 0-N A A1 0-N rA 1-1 R id: rB.B1 rA.A 1-1 rB B1 B1 0-N
Att-ET/inst: Transforming an attribute into an entity type (instance representation). *Inverse:* **ET-Att**	 A A1 A2[a-b] A3 A A1 A3 EA2 A2 id: R.A A2 a-b R 1-1
RT-FK: Transforming a binary relationship type into a foreign key. *Inverse:* **FK-RT**	 A A1 B B1 B2 id: B1 a-b R 0-N A A1 B1[a-b] ref: B1 B B1 B2 id: B1

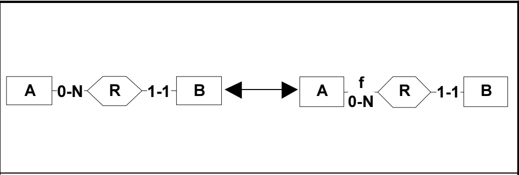
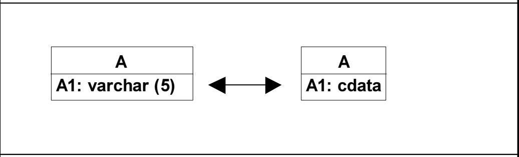
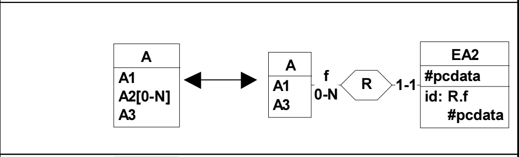
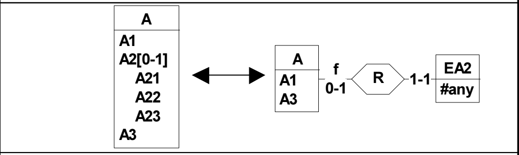
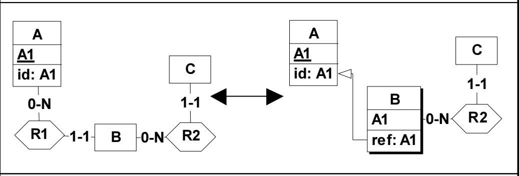
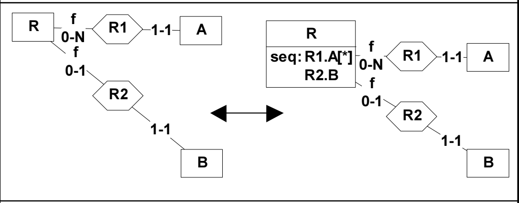
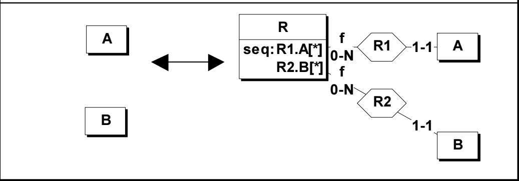
DTD-RT-to-HIER: Transforms a one-to-many (or one-to-one) binary relationship type into a hierarchical relation. The 1-1 role becomes the child role. *Inverse:* **DTD-HIER-to-RT**	
DTD-Att-to-CDATA: Transforms a simple monovalued attribute into an XML CDATA attribute whatever its original type. *Inverse:* **DTD-CDATA-to-Att**	
DTD-Att-to-PCDATA: Transforms a simple attribute into an XML element type with content type PCDATA. *Inverse:* **DTD-PCDATA-to-Att**	 A A1 A2[0-N] A3 A A1 A3 f 0-N R 1-1 EA2 #pcdata id: R.f #pcdata
DTD-Att-to-ANY: Transforms an attribute into an XML element type with content type ANY. *Inverse:* **DTD-ANY-to-Att**	 A A1 A2[0-1] A21 A22 A23 A3 A A1 A3 f 0-1 R 1-1 EA2 #any
DTD-ET-to-ROOT: Gives to any entity type the status of natural root. *Inverse:* **DTD-ROOT-to-ET**	 A A1 id: A1 0-N R1 1-1 B 0-N R2 1-1 C A A1 id: A1 B A1 ref: A1 0-N R2 1-1 C
Create-SEQ-GROUP: Adds a seq group to an entity type. That group contains the child roles played by its children (in an aleatory order). *Inverse:* **Del-SEQ-GROUP**	 R f 0-N R1 1-1 A f 0-1 R2 1-1 B R seq: R1.A[*] R2.B f 0-N R1 1-1 A f 0-1 R2 1-1 B
Create-UNIQUE-ROOT: Adds an unique root to the schema if it possesses multiple potential root entity types. *Inverse:* **Del-ROOT**	 A B R seq: R1.A[*] R2.B[*] f 0-N R1 1-1 A f 0-N R2 1-1 B

Fig. 2. Standard transformations and DTD-specific transformations with their inverse.

The goal of the first two steps is to recover a conceptual view of the legacy database [12] where the entities are interconnected with each other through relationships, while the last two steps belong to the standard forward engineering stream, applied to the XML DTD data model (i.e., converting multiple inter-connected entities into a coherent and hierarchical parent-child structure). All the schema mappings are expressed by means of schema transformation sequences (`PS-to-CS` and `CS-to-DTDs`).

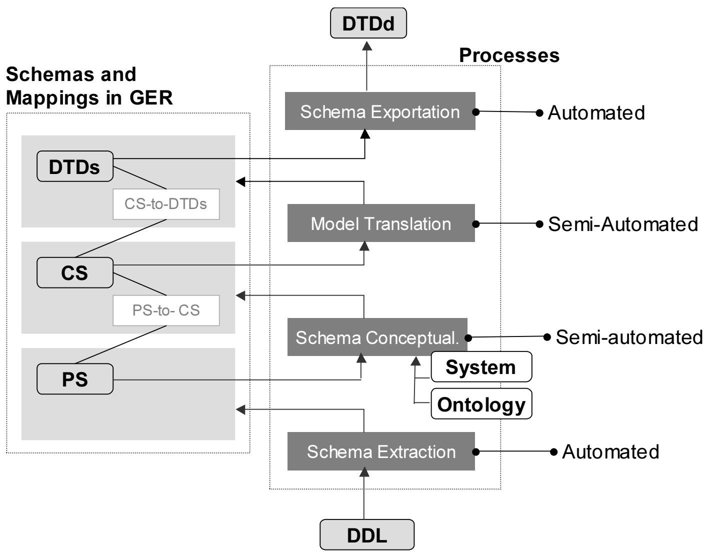


Fig. 3. Database exportation to XML DTD: processes, schemas and transformation sequences.

In the following sections, we describe the first three steps[6] and illustrate them by a small example. We consider an existing database describing aspects of orders. From a technical point of view, the data is managed by an old version of Oracle (Oracle V5), which ignored the concepts of primary and foreign keys.

3.1 Schema Extraction

This phase consists in recovering the (existing) physical schema (`PS`) made up of all the structures and constraints explicitly declared. This process is often easy to automate since it can be carried out by a simple parser which analyses the DDL texts, extracts the data structures and expresses them as the `PS`.
Example. By analyzing the `SQL-DDL` script of the Oracle database, we obtain the physical schema of Figure 4.

[6] We omit the schema exportation process because of its simplicity.

3.2 Schema Conceptualization

Data models of legacy databases cannot express all the semantics of the real world. Limitations of the modelling concepts and information hiding programming practices lead to the incompleteness of the physical schema that only contains the structures *explicitly* expressed in the DDL code, while the other constructs are *implicitly* implemented in the program codes. For example, foreign keys can be explicitly declared in modern relational schemas, but they are implicit (hidden) in obsolete relational schemas (like Oracle V5). This leads to the need to augment the knowledge about the semantics of data, through a *database reverse engineering (DBRE)*, one of the goals of which is to elicit hidden structures and constraints.

To accomplish this, we build on a proven approach, namely the *DB-MAIN DBRE methodology* [12]. The key feature of this approach is threefold. Firstly, all the schemas, whatever their DBMS and their abstraction level, are expressed in the GER. Secondly, it uses the same transformational approach than that of this paper. Thirdly, this approach is supported by an operational CASE tool (Section 4).

Example. The PS of Figure 4 is refined through DBRE in order to recover and interpret the hidden structures and constraints such as foreign keys (interpreted as relationship types), unique keys or enumerated value domains that are unknown in the Oracle V5 model and multivalued attributes that cannot be expressed in the relational model. We thus obtain the conceptual schema (CS) of Figure 4 and the schema transformation sequence (PS-to-CS) that models the DBRE process.

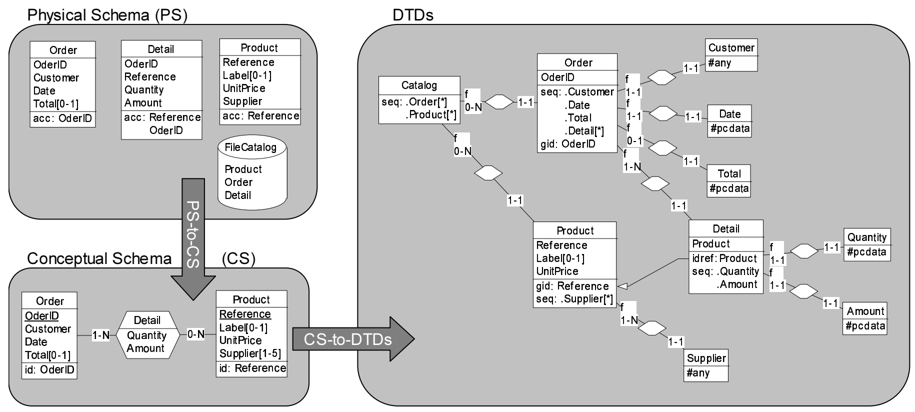


Fig. 4. Example of database exportation: physical, conceptual and DTDs schemas and the transformation sequences between them.

3.3 Model Translation

In this section, we describe a specific case of model translation that is to transform a conceptual schema (`CS`) into a XML DTD (`DTDs`). Model translation is expressed by means of schema transformations and consists, in that case, in applying relevant transformations on the constructs of the `CS` that are not compliant with the XML DTD model. Its execution produces two result types: (1) a target schema expressed in `DTDs`; and (2) a schema transformation sequence `CS-to-DTDs` that is made of all the transformations applied on the source schema to get the target one. This includes standard as well as DTD-specific transformations.

The six steps composing the transformation process are:

1. *Schema preparation.* Invalid complex constructs are automatically cleaned from the source schema.
2. *Hierarchical structure creation.* The user builds a hierarchical organization between entity types by choosing the main concepts and the most natural relations between them according to the modeled application domain.
3. *Role cardinalities extension.* Some father role cardinalities are modified to be valid against the allowed DTD cardinalities.
4. *`gid` and `idref` groups creation.* Identifiers and foreign keys are transformed into `gid` and `idref` groups.
5. *Attributes representation.* Attributes become either XML attributes or XML elements according to the status chosen by the user.
6. *Ordering definition.* Each entity type is given a position relative to its respective parent.

In our approach, model translation is not fully automated and can be considered as a *non-deterministic* process. Indeed, some steps are completely automated while others require some design choices. These user-inputs might have consequences on the properties and the semantic of the resulting schema.

The transformations used in model translation are not necessary semantics-preserving because of the weak expressiveness of DTD XML, notably with regard to the global identifiers or the cardinality constraints.

In the following, we give more details on each step of the transformation process and illustrate them by using the conceptual schema `CS` of Figure 4.

Schema preparation. Schema preparation uses standard schema transformations (Figure 2, first table) to remove invalid constructs such as IS-A relations, n-ary (n>3) and many-to-many relationship types, multivalued and compound attributes.

Example. In Figure 4 (`CS`), the many-to-many relationship type `Detail` is transformed into an entity type (`RT-ET`) and the multivalued attribute `Supplier` of `Product` becomes an entity type (`ATT-ET/instance`).

Hierarchical structure creation. In that step, the structure formed by the entity types and the relationship types is transformed into a tree by choosing the natural root entity type(s), solving the father conflict(s), breaking the cycle(s) and, eventually, adding an unique root. Those four subtasks mainly use two transformations : `RT-to-FK` (a standard transformation) and `DTD-RT-to-HIER`.

`DTD-RT-to-HIER` (Figure 2) is a DTD-specific transformation that is applied on a binary relationship type in order to make it hierarchical. That transformation introduces an explicit notion of superiority (resp. inferiority) between the two concerned entity types according to their role cardinalities. The weak entity type is the one who plays the `1-1` role.

Natural root(s) election. Each entity type representing a significant concept within the modelled domain must be assigned an important status in the schema i.e. an high level in the hierarchy. These meaningful entity types are chosen by the user and are called *natural roots.*
The DTD-specific transformation `DTD-ET-to-ROOT` gives the status of natural root to any entity type in the schema. Let us remark that an entity type that do not play any `1-1` role can not be tied to a father and is consequently a (mandatory) natural root.
For example, in Figure 2 (`DTD-ET-to-ROOT`), the entity type `A` is a mandatory root because it plays no `1-1` role. The entity type `B` is considered important by the user and is therefore subject to the `DTD-ET-to-ROOT` transformation to become a natural root in the target schema.

Father conflicts resolution. A *father conflict* occurs as soon as an entity type has several potential fathers. All father conflicts have to be resolved to get a hierarchical structure. The father conflict resolution consists in choosing one entity type among the potential fathers of the problematic entity type. In practice, the relationship type between the entity type and the elected father is made hierarchical (`DTD-RT-to-HIER`) and all other concerned relationship types become foreign keys (`RT-to-FK`).

Cycles breaking. A cycle is broken by transformation of one of its arcs into a generic foreign key. Breaking a cycle can also be done indirectly by choosing a root entity type among the nodes that compose the cycle.

Unique root addition. If the current schema contains more than one natural root, a new entity type (called *technical root*) is added to compose the unique root of the schema. The DTD-specific transformation `Create-UNIQUE-ROOT` adds that technical root that becomes the father of all the natural roots previously declared or deduced.

Example. The entity types `Order` and `Product` are two mandatory natural roots; no other natural root is elected. The relationship type between `Detail` and `Product` is choosen to become a generic foreign key in order to solve the father conflict that occurs for entity type `Detail`. Finally, an unique technical root (entity type `Catalog`) is added to the schema to get the desired tree organization.

Role cardinalities extension. In `DTDs`, the allowed cardinalities for father roles are `0-N`, `1-N`, `1-1` or `0-1`. In order to respect these limitations, the scope of invalid role cardinalities is extended by applying these two simple principles :

- if `min_card` > `1` then `min_card` = `1`
- if `max_card` > `1` then `max_card` = `N`

That step illustrates the loss of semantics that may occur when a source schema is translated towards a poorer data model.

Example. The father role played by `Product` has cardinalities `1-5`; they are changed into `1-N`.

`gid` and `idref` groups creation. Despite their indisputable dissimilarities, `id` groups composed of one attribute are transformed into `gid` groups. Other `id` groups cannot be translated in `DTDs` and are simply discarded from the schema. In the same way, `ref` groups made up of one attribute only are transformed into (global) `idref` groups even if the scope of the referential constraint is made wider by the transformation.

Example. The `id` groups of entity types `Order` and `Product` are made of a unique attribute; they are transformed into `gid` groups. The `ref` group of entity type `Detail` is renamed `idref` for similar reasons. All other `id` groups are removed from the schema.

Attributes representation. Attributes that do not belong to a `gid` or `idref` group can be handled in three different ways. They can either keep their status of attribute or be promoted into an entity type with simple (`PCDATA`) or tagged (`ANY`) content.

In the first case, the attribute type is simply changed into `cdata` (transformation `DTD-Att-to-CDATA` of Figure 2). In the two latter cases, the attribute is transformed into an entity type (according to an instance representation), a hierarchical relationship type binds both related entity types and the attribute in the leaf entity type is renamed into `#pcdata` or `#any` (`DTD-Att-to-PCDATA` and `DTD-Att-to-ANY` transformations of Figure 2).

Example. All attributes concerned by that transformation become a `PCDATA` or `ANY` entity type except for the attributes `Product.Label` and `Product.UnitPrice` that keep their status. That decision is simply a design choice.

Ordering definition. In the final step, a `seq` group is added to all entity types having more than one children. The current DTD-specific transformation `Create-SEQ-Group` arbitrarily determines the order between the children of a common entity type. This transformation is non-semantic preserving because it adds an explicit notion of order between the different children of an entity type.

Example. `seq` groups have to be added in entity types `Order`, `Detail` and `Reference`.

4 CASE Support

DB-MAIN is a graphical, general-purpose, programmable, CASE environment dedicated to Database Application Engineering. Besides standard functions such as specification entry, examination and management, it includes advanced processors such as transformation toolboxes, reverse engineering processors and schema analysis tools. In particular, DB-MAIN offers a rich set of transformational operators that allow developers to carry out the database exportation in XML DTD in a systematic way. Another interesting feature of DB-MAIN is the Meta-CASE layer, which allows method engineers to customize the tool and to add new concepts, functions, models and even new methods. In particular, DB-MAIN offers a complete development language, *Voyager 2*[13], through which new functions and processors can be developed and seamlessly integrated into the tool. Further details on DB-MAIN can be found in [14].

In the limited scope of this paper, we describe some of the DB-MAIN assistants dedicated to database exportation in XML DTD only.

4.1 Extraction and Exportation Facilities

Database schemas can be extracted by a series of processors. These processors identify and parse the declaration part of the source texts, or analyze catalog tables, and create corresponding abstractions in the repository. Extractors have been developed for SQL, COBOL, CODASYL, IMS, RPG and XML DTD data structures. Additional extractors can be developed easily thanks to the *Voyager 2* environment.

As part of the *Data Migration Project* [15], an XML DTD generator that automatically generates the `DTDd` from a `DTDs` expressed within the GER has been developed. This *Voyager 2* program is based on the specialization rules presented in Section 2.

4.2 Conceptualization and Model Translation

These processes heavily rely on transformation techniques. For some fine-grained reasonings, precise chirurgical transformations have to be carried out on individual constructs. This is a typical way of working in conceptualization activities. In other cases, all the constructs that meet a definite precondition have to be transformed (e.g., all the complex relationship types are transformed in entity types). Finally, some heuristics can be identified and materialized into a transformation plan (high level semi-procedural script that describes how to apply a set of transformations in order to fulfill a particular task or to meet a goal). DB-MAIN offers a dozen predefined model-based transformations including DTD (Section 3.3) translation and untranslation.

5 Conclusions

Recent expansion of the WWW has boosted the emergence of a new generation of applications, combining data and functionalities from various data sources and publishing them coherently in the Web. These applications generally rely on exchanging strongly structured data extracted from databases. Converting from a database model to a semi-structured format often implies a semantic loss, particularly when the meaning of the source data is not well understood, as is usual in legacy databases. Hence the importance of a rich and expressive model to translate the semantics of both database and XML models. As observed in several sections, the translation cannot be lossless due to the weakness of the XML model. Relying on XML Schema instead partially solves the semantic gap. However, a complete translation could only be performed with more powerful XML models that include constraint expression facilities or active components. The approach presented in this paper has put in light the non-deterministic aspect of the model translation process. The same conceptual schema can lead to a large set of equivalent XML structures. Hence the need for more complex methods that take into account other, non functional, criteria for target schema generation. The paper also leaves aside the problem of data generation, that is, the automatic production of XML documents that comply with the DTD that has been computed. This process has been developed and is now supported by the DB-MAIN CASE tool, thanks to the analysis of the history of the transformations used to convert the database schema in XML DTD.

As explained in the introduction, the material developed in this paper is intentionally generic. Considering an ideal scenario according to which the whole contents of an existing (possibly legacy) database is migrated to XML documents, it addresses some of the most critical issues of database-to-XML conversion. More specific, and therefore practical, scenarios, can be derived easily. Database-to-datawarehouse transfer, B2B messages generation, database migration or database-to-Web publishing are some processes that requires strong theoretical bases of similar nature. This paper is a contribution to the development of such bases.

References

1. Florescu, D., Kossmann., D.: Storing and querying XML data using an RDBMS. IEEE Data Engeneering Bulletin **22** (1999) 27–34
2. Carey, M.J., Florescu, D., Ives, Z.G., Lu, Y., Shanmugasundaram, J., Shekita, E.J., Subramanian, S.N.: XPERANTO: Publishing object-relational data as XML. In: WebDB (Informal Proceedings). (2000) 105–110
3. Fernandez, M., Morishima, A., Suciu, D., Tan, W.C.: Publishing relational data in XML: the silkroute approach. (2001)

4. Lee, D., Mani, M., Chiu, F., Chu, W.W.: NeT and CoT: Translating relational schemas to XML schemas using semantic constraints. In: ACM International Conference on Information and Knowledge Management. (2002)
5. Rodriguez-Gianolli, P., Mylopoulos, J.: A semantic approach to XML-based data integration. In: ER Proceedings, Springer-Verlag (2001)
6. Yergeau, F., Cowan, J., Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E.: XML 1.1, W3C recommendation. Technical report, W3C (2004)
7. Hainaut, J.L.: Transformation-based Database Engineering. In: Transformation of Knowledge, Information and Data: Theory and Applications. IDEA Group (2004)
8. Jansen, M., Moller, T.H., Pedersen, T.: Converting XML DTDs to UML diagrams for conceptual data integration. In: Data and Knowledge Engineering. Volume 44. (2003) 323–346
9. Conrad, R., Scheffener, D., Freytag, J.: XML conceptual modeling using UML. In: ER 2000. (2000) 558–571
10. Hainaut, J.L.: Specification preservation in schema transformations - application to semantics and statistics. Data and Knowledge Engineering **11** (1996)
11. Estievenart, F.: DTD-specific transformations - formal description. Technical report, CETIC Research Center, Gosselies, Belgium (2004)
12. Hainaut, J.L.: Introduction to database reverse engineering. Technical report, University of Namur (2002)
13. Englebert, V.: Voyager 2 - reference manual. Technical report, LIBD, University of Namur (2003)
14. Hick, J.M., Englebert, V., Henrard, J., Roland, D., Hainaut, J.L.: The DB-MAIN database engineering CASE tool (version 6.5) - functions overview. DB-MAIN technical manual, Institut d'informatique, University of Namur (2002)
15. Delcroix, C., Thiran, P., Hainaut, J.L.: Approche transformationelle de la reingenierie des donnees. Ingenierie des Systemes d'Information, Hermes, Paris **6** (2001)